Non-linear Second order Abstract Categorial Grammars and deletion

Sylvain Salvati

Non-linear Second order Abstract Categorial Grammars and deletion

Sylvain Salvati

INRIA, LaBRI, Université de Bordeaux

Abstract. We prove that non-linear second order Abstract Categorial Grammars (2ACGs) are equivalent to non-deleting 2ACGs. We prove this result first by using the intersection types discipline. Then we explain how coherence spaces can yield the same result. This result shows that restricting the Montagovian approach to natural language semantics to use only ΛI-terms has no impact in terms of the definable syntax/semantics relations. Moreover from the ACG perspective this result is a generalization of a result by Yoshinaka: it shows that deletion add no expressive power to second order ACGs.

1 Introduction

When defining grammars with parameters, such as Macro Grammars [Fis68], Parallel Multiple Context-Free Grammars [SMFK91], Higher-order macro grammars [Dam82], one can duplicate or even not use some of those parameters in the course of the derivation. While it is clear that duplication is a necessary feature for the expressiveness of those formalisms, as otherwise the higher-order hierarchy collapses (see [Sal07]), it is less clear what the status of deletion is. Fischer [Fis68] showed that for IO grammars deletion does not augment the expressive power, but it has been shown by Leguy that for OI grammars deletion is necessary to generate the full class of languages [Leg81]. Non-deleting grammars often simplify theoretical investigations about languages definable with a given class of grammars, but they also make it easier to address algorithmic problems (like parsing) related to those grammars. Under those considerations, it is useful to know whether the assumption of working with non-deleting grammars can be used for free.

As showed in [KS14], Second order Abstract Categorial Grammars [dG01], in particular when the restriction of being linear[1] is lifted, are a generalization of higher-order IO macro grammars [Dam82] which define languages of strings and possibly of trees, to languages of simply typed λ-terms. They are also a generalization in the sense that they do not assume a certain restriction that is implicit in Damm's definition and which is called *safety* [BO09]. For this class of grammars, it is less clear than for IO macro grammars whether deletion is not

[1] Here *linear* means *non-duplicating* according to the usual terminology of formal language theory. We employ the term *linear* for historical reasons, and because of its connection with linear logic.

contributing to the expressive power. In particular, the generated language may very well contain λ-terms which contain vacuously bounded variables such as in the term $\exists(\lambda x.\top)$. There is nevertheless some hint in the literature with the result of Yoshinaka [Yos06] which shows that deletion is superfluous when duplication is not allowed. The way Yoshinaka treats vacuous abstraction consists in constructing a non-deleting grammar whose language is the set of all the λ-terms of the language of the initial grammar that do not contain vacuous abstractions. We here generalize Yoshinaka's result in two directions, first we prove it without the assumption that the grammar is not duplicating, second, we treat vacuous abstractions in a more general manner: instead of generating the language directly, we generate a language whose elements are representations of the terms in the initial language but with the property that these representations do not contain any vacuous abstraction. Then we can either map each representation to the represented terms without appealing to deletion, or we may filter out the terms that contain vacuous abstraction using simple finite state techniques and obtain the same result as Yoshinaka.

We propose two methods so as to prove this result. The first approach is syntactic and uses a form of intersection typing without weakening rule. This approach is in line with the treatment of parsing non-linear second order ACGs proposed in [Sal10]. The second method elaborates on the more semantic approach proposed in [KS14] and relates the result to stable functions introduced by Berry [Ber78] via coherence spaces, their presentation as a model of linear logic proposed by Girard [Gir86]. We believe that the second approach, even though less intuitive at first sight is more elegant and more satisfactory.

2 Typing and deletion

Simple types over the finite set of atoms $\mathcal{A}$ are inductively defined by:

$$\mathcal{T}(\mathcal{A}) ::= \mathcal{A} \mid (\mathcal{T}(\mathcal{A}) \to \mathcal{T}(\mathcal{A}))$$

The order $\mathtt{ord}(\alpha)$ of a type α in $\mathcal{T}(\mathcal{A})$ is 0 when α is in $\mathcal{A}$ and it is $\max(\mathtt{ord}(\alpha_1)+1, \mathtt{ord}(\alpha_2))$ when $\alpha = \alpha_1 \to \alpha_2$.

A Higher Order Signature (HOS) Σ is a triple $(\mathcal{A}, \mathcal{C}, \tau)$ so that:

1. $\mathcal{A}$ is a finite set of atoms,
2. $\mathcal{C}$ is a finite set of constants,
3. τ is a function from $\mathcal{C}$ to $\mathcal{T}(\mathcal{A})$.

In general we write $\mathcal{T}(\Sigma)$ for $\mathcal{T}(\mathcal{A})$. The order $\mathtt{ord}(\Sigma)$ of a HOS Σ is $\max\{\mathtt{ord}(\tau(c)) \mid c \in \mathcal{C}\}$.

We assume that for each HOS Σ we have an infinite countable set of typed λ-variables of the form x^α with α in $\mathcal{T}(\Sigma)$. We define the sets of typed λ-terms over Σ, $(\Lambda^\alpha_\Sigma)_{\alpha\in\mathcal{T}(\Sigma)}$ as the smallest sets such that:

1. x^α is in Λ^α_Σ,
2. c is in $\Lambda^{\tau(c)}_\Sigma$,

3. if $M \in \Lambda_\Sigma^{\alpha\to\beta}$, $N \in \Lambda_\Sigma^\alpha$, then $(MN) \in \Lambda_S^\beta$
4. if $M \in \Lambda_\Sigma^\beta$, then $\lambda x^\alpha.M \in \Lambda_\Sigma^{\alpha\to\beta}$.

The set of ΛI-terms is defined $(\Lambda I_\Sigma^\alpha)_{\alpha\in\mathcal{T}(\Sigma)}$ as the smallest sets such that:

1. x^α is in ΛI_Σ^α,
2. c is in $\Lambda I_\Sigma^{\tau(c)}$,
3. if $M \in \Lambda I_\Sigma^{\alpha\to\beta}$, $N \in \Lambda I_\Sigma^\alpha$, then $(MN) \in \Lambda I_S^\beta$
4. if $M \in \Lambda I_\Sigma^\beta$ and $x^\alpha \in FV(M)$, then $\lambda x^\alpha.M \in \Lambda I_\Sigma^{\alpha\to\beta}$.

We define sorted intersection types, $(\mathcal{I}_\alpha)_{\alpha\in\mathcal{T}(\mathcal{A})}$ as follows:

$$\begin{aligned}\mathcal{I}_\alpha &::= \alpha \text{ when } \alpha \in \mathcal{A}\\ \mathcal{I}_{\alpha\to\beta} &::= \{p_1;\ldots;p_n\} \to p \mid \omega_\alpha \to p \text{ with } p \in \mathcal{I}_\beta,\ n > 0,\ p_1,\ldots,p_n \in \mathcal{I}_\alpha\end{aligned}$$

Note that $\mathcal{I}_\alpha$ is finite for every α.

We also define two subfamilies of $(\mathcal{I}_\alpha)_{\alpha\in\mathcal{T}(\Sigma)}$, $(\mathcal{I}_\alpha^+)_{\alpha\in\mathcal{T}(\Sigma)}$, and $(\mathcal{I}_\alpha^-)_{\alpha\in\mathcal{T}(\Sigma)}$:

$$\begin{aligned}\mathcal{I}_\alpha^+ &::= \alpha \text{ when } \alpha \in \mathcal{A}\\ \mathcal{I}_\alpha^- &::= \alpha \text{ when } \alpha \in \mathcal{A}\\ \mathcal{I}_{\alpha\to\beta}^+ &::= \{p_1;\ldots;p_n\} \to p \mid \omega_\alpha \to p \text{ with } p \in \mathcal{I}_\beta^+,\ n > 0,\ p_1,\ldots,p_n \in \mathcal{I}_\alpha^-\\ \mathcal{I}_{\alpha\to\beta}^- &::= \{q\} \to p \text{ with } q \in \mathcal{I}_\alpha^+ \text{ and } p \in \mathcal{I}_\beta^-\end{aligned}$$

Typing environments Γ, are functions that associate a subset of $\mathcal{I}_\alpha$ to variables like x^α. The typing environments Γ we consider are finite in the sense that the set of variables x^α such that $\Gamma(x^\alpha)$ is not the empty set is finite. This set of variables is the *domain* of Γ, and is written $\mathrm{Dom}(\Gamma)$. Given two typing environment Γ and Δ, we write $\Gamma \cup \Delta$ for the environment such that $(\Gamma \cup \Delta)(x^\alpha) = \Gamma(x^\alpha) \cup \Delta(x^\alpha)$. We may write environment Γ as a sequence $x_1 : p_{1,1},\ldots,x_1 : p_{1,k_1},\ldots,x_n : p_{n,1},\ldots,x_n : p_{n,k_n}$, meaning that $\mathrm{Dom}(\Gamma) = \{x_1;\ldots;x_n\}$ and $\Gamma(x_i) = \{p_{i,1};\ldots;p_{i,k_i}\}$.

The typing system is given by the following inference rules:

Rules	Notation
$\dfrac{p \in \mathcal{I}_\alpha}{x^\alpha : p \vdash x^\alpha : p}$	(x^α, p)
$\dfrac{p \in \mathcal{I}^-_\alpha}{\vdash c : p}$	(c, p)
$\dfrac{d :: \Gamma \vdash M : \omega_\alpha \to p \quad N \in \Lambda^\alpha_\Sigma}{\Gamma \vdash (MN) : p}$	$app_\omega(d, N)$
$\dfrac{d_0 :: \Gamma_0 \vdash M : \{p_1; \ldots; p_n\} \to p \quad \forall i \in [1; n] d_i :: \Gamma_i \vdash N : p_i}{\Gamma_0 \cup \Gamma_1 \cup \ldots \cup \Gamma_n \vdash (MN) : p}$	$app_I(d_0, d_1, \ldots, d_n)$
$\dfrac{d :: \Gamma \vdash M : p \quad x^\alpha \notin \mathtt{Dom}(\Gamma)}{\Gamma \vdash \lambda x^\alpha.M : \omega^\alpha \to p}$	$\Lambda_\omega x^\alpha.d$
$\dfrac{d :: \Gamma \cup \{x^\alpha : p_1, \ldots, x^\alpha : p_n\} \vdash M : p \quad x^\alpha \notin \mathtt{Dom}(\Gamma)}{\Gamma \vdash \lambda x^\alpha.M : \{p_1; \ldots; p_n\} \to p}$	$\Lambda_I x^\alpha.d$

Since in the sequel we need to manipulate derivations, the figures gives a notation for the derivation trees of the typing system. When we write $d :: \Gamma \vdash M : p$, we mean that d is a derivation tree of $\Gamma \vdash M : p$.

Definition 1 (substitution). *A* derivation substitution σ*, is a partial function that associates to a variable* x^α *a term* N *of type* α *and to pairs* (x^α, p) *where* $p \in I^\alpha$ *a derivation* $d :: \Gamma \vdash N : p$*. Moreover whenever* σ *is defined for a pair* (x^α, p) *then it is also defined for the variable* x^α *and the subject of the derivation* $\sigma(x^\alpha, p)$ *is the term* $\sigma(x^\alpha)$*. A derivation substitution, induces naturally a substitution* $\overline{\sigma}$*.*

Given a derivation $d :: \Gamma \vdash M : p$ *and a derivation substitution* σ *such that for all* $x : p$ *in* Γ*, if* $\sigma(x)$ *is defined, then* $\sigma(x, p)$ *is defined, we inductively define* $d.\sigma$ *as the derivation obtained as follows:*

1. *if* $d = (x^\alpha, p_i)$*, and* $\sigma(x^\alpha)$ *is defined, then* $d.\sigma = \sigma(x^\alpha, p_i)$*,*
2. *if* $d = (y^\beta, q)$ *and* $\sigma(y^\beta)$ *is undefined then* $d.\sigma = d$*,*
3. *if* $d = (c, p)$ *then* $d.\sigma = d$*,*
4. *if* $d = app_\omega(d', N)$ *then* $d.\sigma = app_\omega(d'.\sigma, N.\sigma)$*,*
5. *if* $d = app_I(d_0, d_1, \ldots, d_n)$ *then* $d.\sigma = app_I(d_0.\sigma, d_1.\sigma, \ldots, d_n.\sigma)$*,*
6. *if* $d = \Lambda_\omega y^\beta.d'$ *and* $\sigma(y^\beta)$ *is undefined, then* $d.\sigma = \Lambda_\omega y^\beta.d'.\sigma$*,*
7. *if* $d = \Lambda_I y^\beta.d'$ *and* $\sigma(y^\beta)$ *is undefined, then* $d.\sigma = \Lambda_I y^\beta.d'.\sigma$*,*

For the case of λ*-abstraction when* σ *is defined for finitely many variables, using* α*-conversion, we can always satisfy the condition that* $\sigma(y^\beta)$ *is undefined.*

Lemma 1 (substitution). *If* $d :: \Gamma, x^\alpha : p_1 \ldots, x^\alpha : p_n \vdash M : p$ *(with* $x^\alpha \notin \mathtt{Dom}(\Gamma)$*) and for all* i *in* $[n]$*,* $d_i :: \Gamma_i \vdash N : p_i$ *then* $d' :: \Gamma, \Gamma_1, \ldots, \Gamma_n \vdash M[x^\alpha := N] : p$ *with* $d' = d.\sigma$ *and* $\sigma(x^\alpha, p_i) = d_i$*.*

Proof. This Lemma can easily be proved by induction on the structure of d.

We now define a relation of reduction on derivation.

Definition 2. *Given a derivation $d :: \Gamma \vdash M : p$ such that $M = C[(\lambda x^\alpha.M_1)M_2]$, then we define the derivation $(d \downarrow C[])$ as follows:*

1. $(app_\omega(\Lambda_\omega x^\alpha.d', M_2) \downarrow []) = d'[x^\alpha := M_2]$
2. $(app_I(\Lambda_I x^\alpha.d_0, d_1, \ldots, d_n) \downarrow []) = (d \downarrow C[]) = d_0.\sigma$ *with* $\sigma(x^\alpha, p_i) = d_i$ *when* $d_0 :: \Gamma \vdash M_1 : \{p_1; \ldots; p_n\} \to p$,
3. $(app_\omega(d', N) \downarrow M'C'[]) = app_\omega(d', N')$ *with* $N' = C'[M_1[x^\alpha := M_2]]$
4. $(app_I(d_0, d_1, \ldots, d_n) \downarrow M'C'[]) = app_I(d_0, (d_1 \downarrow C'[]), \ldots, (d_n \downarrow C'[]))$.
5. $(app_\omega(d', N) \downarrow C'[]M') = app_\omega((d' \downarrow C'[]), N)$
6. $(app_I(d_0, d_1, \ldots, d_n) \downarrow C'[]M') = app_I((d_0 \downarrow C'[]), d_1, \ldots, d_n)$
7. $(\Lambda_\omega y^\beta.d' \downarrow \lambda y^\beta.C'[]) = \Lambda_\omega y^\beta.(d' \downarrow C'[])$
8. $(\Lambda_I y^\beta.d' \downarrow \lambda y^\beta.C'[]) = \Lambda_I y^\beta.(d' \downarrow C'[])$

Lemma 2 (β-contraction). *Given a derivation $d :: \Gamma \vdash M : p$ such that $M = C[(\lambda x^\alpha.M_1)M_2]$, then $(d \downarrow C[]) :: \Gamma \vdash C[M_1[x^\alpha := M_2]] : p$.*

Proof. Simple induction on $C[]$.

So now in case $d :: \Gamma \vdash M : p$, $M = C[(\lambda x^\alpha.M_1)M_2]$ and $N = C[M_1[x^\alpha := M_2]]$ we call $(d \downarrow C[])$ the derivation induced by the β-contraction of M into N.

Theorem 1. *For all M, N in $\Lambda^\alpha_{\Sigma_\omega}$, $M =_\beta N$ implies that for all environment Γ and p in $\mathcal{I}_\alpha$, we have:*

$$\Gamma \vdash M : p \text{ iff } \Gamma \vdash N : p$$

Proof. This Theorem is a special instance of the β-conversion Theorem for intersection types in the λ-calculus. In the special case of such a typing see [Sal10].

We now turn to a particular technical Lemma that shows the interest of $\mathcal{I}^+_\alpha$ and $\mathcal{I}^-_\alpha$ with respect to that typing system. A typing environment is said negative when for every x^α, $\Gamma(x^\alpha)$ is included in $\mathcal{I}^-_\alpha$.

Lemma 3. *Given M in Λ^α_Σ and in η-long form, there is a unique negative environment Γ and a unique p in $\mathcal{I}^+_\alpha$ such that $\Gamma \vdash M : p$.*

Proof. From Theorem 1, we may assume that M is in β-normal form.

We proceed by induction on the structure of M.

In case $M = \lambda x^\alpha.M'$, the conclusion follows immediately from the induction hypothesis.

In case $M = hM_1 \ldots M_n$ with h being either a constant or a variable of type $\alpha_1 \to \cdots \to \alpha_n \to \beta$ (with β in $\mathcal{A}$), then by induction hypothesis, for each i, there is a unique negative environment Γ_i and a unique p_i in $\mathcal{I}^+_\mathcal{A}$ such that $\Gamma \vdash N_i : p_i$. In case h is a variable x^α, the only possible type for x^α is $\{p_1\} \to \cdots \to \{p_n\} \to \beta$. Similarly in case h is a constant c, the only possible typing for c is $\{p_1\} \to \cdots \to \{p_n\} \to \beta$.

A derivation $d :: \Gamma \vdash M : p$ is said *faithful* when Γ is negative and p is in $\mathcal{I}^+_\alpha$. The previous lemma shows that each term has a unique faithful derivation. We now take advantage of this derivation so as to map each λ-term to an equivalent λI-term.

Given a type p in $\mathcal{I}_\alpha$ we map it to a type $(\!|p|\!)_\alpha$ of $\mathcal{T}(\Sigma)$ as follows:

1. if α is in $\mathcal{A}$, $(\!|p|\!)_\alpha = \alpha$
2. $(\!|\omega_\alpha \to p|\!)_{\alpha\to\beta} = (\!|p|\!)_\beta$
3. $(\!|\{p_1;\ldots;p_n\} \to p|\!)_{\alpha\to\beta} = (\!|p_1|\!)_\alpha \to \cdots \to (\!|p_n|\!)_\alpha \to (\!|p|\!)_\beta$

For $(\!|\cdot|\!)_\alpha$ to be functional we implicitly assume without loss of generality that $\mathcal{I}_\alpha$ is totally ordered and that when we write $\{p_1;\ldots;p_n\}$, p_i is smaller than p_{i+1} for that order.

We now define the signature $\Sigma_w = (\mathcal{A}, \mathcal{C}_w, \rho)$ such that, $\mathcal{C}_w = \{\langle c, p\rangle \mid c \in \mathcal{C} \wedge p \in I^-_{\tau(c)}\}$ and $\rho(\langle c, p\rangle) = (\!|p|\!)_{\tau(c)}$. Given an environment Γ, a *variable interpretation* of Γ, ν, is an injective function that assosiates a variable $z^{(\!|p|\!)_\alpha}$ to every pair (x^α, p) when p in $\Gamma(x^\alpha)$. Given a derivation d of the sequent $\Gamma \vdash M : p$, with p in $\mathcal{I}_\alpha$, and a variable interpretation ν of Γ we define $(\!|d|\!)_\nu$ to be a λI-term of type $(\!|p|\!)_\alpha$ as follows:

1. if $d = (x^\alpha, p_i)$, then $(\!|d|\!)_\nu = \nu(x^\alpha, p_i)$,
2. if $d = (c, p)$ then $(\!|d|\!)_\nu = \langle c, p\rangle$,
3. if $d = app_\omega(d', N)$ then $(\!|d|\!)_\nu = (\!|d'|\!)_\nu$,
4. if $d = app_I(d_0, d_1, \ldots, d_n)$ then $(\!|d|\!)_\nu = (\!|d_0|\!)_\nu (\!|d_1|\!)_\nu \ldots (\!|d_n|\!)_\nu$,
5. if $d = \Lambda_\omega x^\alpha.d'$, $(\!|d|\!)_\nu = (\!|d'|\!)_\nu$,
6. if $d = \Lambda_I x^\alpha.d'$, $(\!|d|\!)_\nu = \lambda z_1^{(\!|p_1|\!)_\alpha} \ldots z_n^{(\!|p_n|\!)_\alpha}.(\!|d'|\!)_{\nu'}$, when $d :: \Gamma \vdash M : \{p_1, \ldots, p_n\} \to p$ and where ν' is equal to ν on its domain and maps the pairs (x^α, p_i) to the fresh variables $z_i^{(\!|p_i|\!)_\alpha}$.

It can be easily checked that, for M in Λ^α_Σ $(\!|d :: \Gamma \vdash M : p|\!)_\alpha$ is in $\Lambda I^{(\!|p|\!)_\alpha}_{\Sigma_w}$.

Lemma 4. *If M is in Λ^α_Σ and $d :: \Gamma \vdash M : p$ then $(\!|d|\!)_\nu$ is in $\Lambda I^{(\!|p|\!)_\alpha}_{\Sigma_w}$.*

Interestingly $(\!|\cdot|\!)$ commutes with substitution.

Lemma 5. *Given $d :: \Gamma, x^\alpha : p_1, \ldots, x^\alpha : p_n \vdash M : p$ and for all i in $[n]$, $d_i :: \Gamma_i \vdash N : p_i$, $(\!|d[(x^\alpha, p_i) := d_i]|\!)_\nu = (\!|d|\!)_{\nu'}[\nu'(x^\alpha, p_i) := (\!|d_i|\!)_\nu]_{i\in[1,n]}$ where ν' is equal to ν on its domain and is mapping (x^α, p_i) to some fresh variables.*

Proof. We proceed by induction on the structure of d.

Lemma 6. *Given $d :: \Gamma \vdash M : p$ and $d' :: \Gamma \vdash N : p$ two derivations such that $d \to_\beta d'$, then $(\!|d|\!)_\nu \xrightarrow{*}_\beta (\!|d'|\!)_\nu$.*

Proof. It is a simple induction on the context wrapping the redex of M that is contracted to obtain N. Notice that the reduction from $(\!|d|\!)_\nu$ to $(\!|d'|\!)_\nu$ may require more than one step. Indeed, the translation $(\!|\cdot|\!)$ transform one redex of M into n redices in $(\!|d|\!)$ where n corresponds to the number of typing judgments that redex is the subject of in d.

Theorem 2. *If $M =_\beta N$, $d :: \Gamma \vdash M : p$ then there is $d' :: \Gamma \vdash N : p$ and $(\!|d|\!)_\nu =_\beta (\!|d'|\!)_\nu$.*

Proof. Since $M =_\beta N$ and $d :: \Gamma \vdash M : p$ then, by Theorem 1 there is $d' :: \Gamma \vdash N : p$. The fact that $(\!|d|\!)_\nu =_\beta (\!|d'|\!)_\nu$ is just a recursive application of Lemma 6 on the reduction sequences that normalize N and M.

Now we are going to give a pair of translations that are used as inverse of the transformation $(\!|\cdot|\!)$ on faithful derivations. The first tranformation is $\langle p, \alpha, M\rangle^+$ where p is in $\mathcal{I}^+_\alpha$ and M is in $\Lambda^{(\!|p|\!)_\alpha}$ and that produces a term of type α, while the second transformation $\langle p, \alpha, M\rangle^-$ with p is in $\mathcal{I}^-_\alpha$ and M is in Λ^α produces a term of type $(\!|p|\!)_\alpha$; they are mutually recursively defined by:

1. $\langle p, \alpha, M\rangle^+ = M$ if α is atomic,
2. $\langle \{p_1; \ldots; p_n\} \to p, \alpha \to \beta, M\rangle^+ = \lambda x^\alpha.\langle p, \beta, M\langle p_1, \alpha, x^\alpha\rangle^- \ldots \langle p_n, \alpha, x^\alpha\rangle^-\rangle^+$
3. $\langle \omega_\alpha \to p, \alpha \to \beta, M\rangle^+ = \lambda x^\alpha.\langle p, \beta, M\rangle^+$
4. $\langle p, \alpha, M\rangle^- = M$ if α is atomic,
5. $\langle \{q\} \to p, \alpha \to \beta, M\rangle^- = \lambda x^{(\!|q|\!)_\alpha}.\langle p, \beta, M\langle q, \alpha, x^{(\!|q|\!)_\alpha}\rangle^+\rangle^-$

Notice that obviously if M is in $\Lambda^{(\!|q|\!)_\alpha}$, then $\langle q, \alpha, x^{(\!|q|\!)_\alpha}\rangle^+[x^{(\!|q|\!)_\alpha} := M] = \langle q, \alpha, M\rangle^+$.

Lemma 7. *For every faithful derivation $d :: \Gamma \vdash M : p$, every variable substitution ν, σ and h so that:*

- *σ is a substitution that maps a variable $z^{(\!|p|\!)_\alpha}$ to $\langle p', \alpha, x^\alpha\rangle^-$ when $\nu(x^\alpha, p') = z^{(\!|p|\!)_\alpha}$,*
- *h is the homomorphism that maps every constant $\langle c, p\rangle$ to the term $\langle p, \tau(x), c\rangle^-$,*

we have:

$$\langle p, \alpha, h((\!|d|\!)_\nu).\sigma\rangle^+ =_\beta M$$

Proof. Theorem 2 allows us to consider that, without loss of generality, M is in long normal form. Then the proof is done by a simple induction on the structure of M.

3 Wrapping up with grammars

Given a second order (non-linear) ACG $G = (\Theta, \Sigma, \mathcal{L}, S)$ (we assume without loss of generality that $\mathcal{L}(S)$ is atomic), we construct the grammar $G_\omega = (\Theta', \Sigma_\omega, \mathcal{L}_\omega, [S, \mathcal{L}(S)])$, where:

1. $\Theta = (\mathcal{A}, \mathcal{C}, \tau)$ and $\Theta' = (\mathcal{A}', \mathcal{C}', \rho)$ so that $\mathcal{A}' = \{[A, p] \mid A \in \mathcal{A} \wedge p \in I_{\mathcal{L}(A)}\}$, $\mathcal{C}' = \{[c, d] \mid d ::\vdash \mathcal{L}(c) : p\}$ and $[c, d] = [A_1, p_{1,1}] \to \cdots [A_1, p_{1,k_1}] \to \cdots \to [A_n, p_{n,1}] \to \cdots \to [A_n, p_{n,k_n}] \to [A, q]$ when $\tau(c) = A_1 \to \cdots \to A_n \to A$ and $d ::\vdash \mathcal{L}(c) : P_1 \to \cdots \to P_n \to q$ with $P_i = \{p_{i,1}; \ldots; p_{i,k_i}\}$,
2. $\mathcal{L}_w([c, d]) = (\!|d|\!)$.

A simple induction shows that $\{\mathcal{L}_\omega(M) \mid M \text{ closed and } M \in \Theta_\omega^{[A,p]}\}$ is equal to the set $\{(\!|d|\!) \mid \exists M \in \Theta^A.d ::\vdash \mathcal{L}(M) : p\}$. Furthermore, Lemma 7 gives that $\mathcal{L}(G) = h(\mathcal{L}_\omega(G))$ this shows that when $\mathcal{L}(G)$ is a string or a tree language then $\mathcal{L}(G) = \mathcal{L}_\omega(G)$ (since in that case h is a bijective relabeling), and that in the other cases the terms of the original language can be read from the terms in $\mathcal{L}_\omega(G)$ by composing it with h.

If we define the types $\mathcal{R}_\alpha$ by induction on α to be $\mathcal{R}_\alpha = \alpha$ when α is atomic, and $\mathcal{R}_{\alpha\to\beta} = \{\mathcal{R}_\alpha\} \to \mathcal{R}_\beta$, then the constants of the form $\langle c, \mathcal{R}_{\tau(c)}\rangle$ are mapped by h to linear λ-terms. Using simple Scott models as in [KS14] one can recognize the set of terms that contain only constants of the form $\langle c, \mathcal{R}_{\tau(c)}\rangle$. Then, with usual constructs, we can restrict the grammar to the sub-language of terms that contain only such constants. Applying h to that language yields the language of terms of the initial grammars that did not contain any vacuous abstraction. This simple adaptation allows us to generalize Yoshinaka's result in [Yos06].

4 Connection with linear logic

The ideas developed in section 3 are rather simple: take a type system without weakening and then see how it connects with β-reduction so as to use its properties to remove weakening. Most of the effort is dedicated to establish the consistency of typing and typing derivations with β-reduction. We would feel somewhat better if we could take those results off the shelf. This is actually possible by using tools coming from denotational semantics. It suffices to use stable functions introduced by Berry [Ber78] to achieve this. Somehow to make the connection it is easier to use their presentation under the guise of coherence spaces that are at the origin of linear logic [Gir86].

We quickly recall what coherence spaces are. A coherence space C, is a pair $(|C|, \frown)$ where $|C|$ is a set (here it will always be finite), the *web*, and $\frown$ is a symmetric and reflexive relation on $|C|$. We denote by $\asymp$ the reflexive closure of the complement of $\frown$, so $C^\perp = (|C|, \asymp)$ is also a coherence space. A subset X of $|C|$ is called a *state* when for every x, y in X, $x \frown y$.

There are other operations that are useful to construct coherence spaces such as $\multimap$ and !. Given two coherence spaces $C_1 = (|C_1|, \frown_1)$ and $C_2 = (|C_2|, \frown_2)$, the coherence space $C_1 \multimap C_2$ is the coherence space $(|C_1| \times |C_2|, \frown)$ where $(a_1, b_1) \frown (a_2, b_2)$ when $a_1 \frown a_2$ implies $b_1 \frown b_2$ and $b_1 \asymp b_2$ implies $a_1 \asymp a_2$. Given a coherence space $C = (|C|, \frown)$, $!C = (|!C| \frown_!)$ is the coherence space where $|!C|$ is the set of states of C and $X \frown_! Y$ when $X \cup Y$ is in $|!C|$.

Now if at each type α we associate a coherence space C_α as follows, if α is atomic, then we let C_α be the coherence space with only one point (noted $\bullet$ below), and $C_{\alpha\to\beta} = !C_\alpha \multimap C_\beta$. Now the important point to raise here is that the families of intersection types $(\mathcal{I}^+_\alpha)_{\alpha\in\mathcal{T}(\Sigma)}$, and $(\mathcal{I}^-_\alpha)_{\alpha\in\mathcal{T}(\Sigma)}$ can be represented in C_α. For this we define $\mathcal{C}^+_\alpha$ and $\mathcal{C}^-_\alpha$ by induction on α following the definitions

we have given for the intersection types:

$$\begin{aligned}
\mathcal{C}^+_\alpha &::= \{\bullet\} \text{ when } \alpha \in \mathcal{A} \\
\mathcal{C}^-_\alpha &::= \{\bullet\} \text{ when } \alpha \in \mathcal{A} \\
\mathcal{C}^+_{\alpha\to\beta} &::= \{(D,p) \mid D \subseteq \mathcal{C}^-_\alpha \wedge p \in \mathcal{C}^+_\alpha\} \\
\mathcal{C}^-_{\alpha\to\beta} &::= \{(\{q\},p) \mid q \in \mathcal{C}^+_\alpha \wedge p \in \mathcal{C}^-_\beta\}
\end{aligned}$$

We need to prove that $\mathcal{C}^+_\alpha$ and $\mathcal{C}^-_\alpha$ is included in $|C_\alpha|$. For this we need the following lemma.

Lemma 8. *For every α, $\mathcal{C}^+_\alpha \cup \mathcal{C}^-_\alpha \subseteq |C_\alpha|$ and if $p_1, p_2 \in \mathcal{C}^+_\alpha$ and $n_1, n_2 \in \mathcal{C}^-_\alpha$ then: $p_1 \asymp p_2$ and $n_1 \frown n_2$.*

Proof. Simple induction on α.

Now, as coherence spaces form a model of linear logic, the co-Kleisly construction gives a model of simply typed λ-calculus where each closed term of type α is interpreted as a state of C_α. Evaluating a term in this model amounts to taking a valuation that maps variables of type α to element in $!C_\alpha$, and interpret terms of type α into $!C_\alpha$, then the interpretation of terms is given by induction as follows:

1. $[\![x]\!]_\nu = \nu(x)$,
2. $[\![MN]\!]_\nu = [\![M]\!]_\nu * [\![N]\!]_\nu$ where $R_1 * R_2 = \{p \mid \exists (P,p) \in R_1, P \subseteq R_2\}$,
3. $[\![\lambda x^\alpha.M]\!]_\nu = \{(P, [\![M]\!]_{\nu[x^\alpha := P]}) \mid P \in !C_\alpha\}$,
4. $[\![c]\!]_\nu = \mathbf{c}_{\tau(c)}$

where $\mathbf{c}_\alpha = \{\bullet\}$ when α is atomic and $\mathbf{c}_{\alpha\to\beta} = \{(\{\mathbf{c}_\alpha\}, \mathbf{c}_\beta)\}$ otherwise.

It remains to show that the interpretation of each closed term contains an element in $\mathcal{C}^+_\alpha$. This element is unique according to Lemma 8, so this gives an alternate proof of Lemma 3. Now using the fact that when $P_1 \cup P_2$ is a state then $R * (P_1 \cap P_2) = (R * P_1) \cap (R * P_2)$, which is just a respelling of conditional multiplicativity (that is equivalent to stability in ω-algebraic CPOs), we can define for a given R and P so that $Q \subseteq R * P$ the least $R' \subseteq R$ and the least $P' \subseteq P$ so that $Q \subseteq R' * P'$. Now given a term M so that $p \in \{M\}_\nu$ for some p in $\mathcal{C}^+_\alpha$, we can define the least substitution ν' so that $p \in \{M\}_{\nu'}$ and for every x, $\nu'(x) \subseteq \nu(x)$. By induction, we can then *specialize* each term to the set of elements in their semantics in a similar way as we did in section 2. We will have as a bonus that the transformation is semantic invariant and moreover we will not have to prove the theorems concerning correctness of the typing system with respect to β-reduction as they come for free from the theory of linear logic.

References

Ber78. Gérard Berry. Stable models of typed lambda-calculi. In *Proceedings of the Fifth Colloquium on Automata, Languages and Programming*, pages 72–89, London, UK, UK, 1978. Springer-Verlag.

BO09. Willam Blum and C.-H. Luke Ong. The safe lambda calculus. *Logical Methods in Computer Science*, 5(1:3):1–38, 2009.

Dam82. Werner Damm. The IO- and OI-hierarchies. *Theoretical Computer Science*, 20:95–207, 1982.

dG01. Philippe de Groote. Towards abstract categorial grammars. In Association for Computational Linguistic, editor, *Proceedings 39th Annual Meeting and 10th Conference of the European Chapter*, pages 148–155. Morgan Kaufmann Publishers, 2001.

Fis68. Michael J. Fischer. *Grammars with macro-like productions.* PhD thesis, Harvard University, 1968.

Gir86. Jean-Yves Girard. The system F of variable types, fifteen years later. *Theor. Comput. Sci.*, 45(2):159–192, 1986.

KS14. Gregory M. Kobele and Sylvain Salvati. The IO and OI hierarchies revisited. *Information and Computation*, (0):–, 2014.

Leg81. Bernard Leguy. Grammars without erasing rules. the OI case. In Egidio Astesiano and Corrado Böhm, editors, *CAAP '81*, volume 112 of *Lecture Notes in Computer Science*, pages 268–279. Springer Berlin Heidelberg, 1981.

Sal07. Sylvain Salvati. Encoding second order string acg with deterministic tree walking transducers. In S. Wintner, editor, *Proceedings FG 2006: the 11th conference on Formal Grammars*, FG Online Proceedings, pages 143–156. CSLI Publications, 2007.

Sal10. Sylvain Salvati. On the membership problem for non-linear ACGs. *Journal of Logic Language and Information*, 19(2):163–183, 2010.

SMFK91. Hiroyuki Seki, Takashi Matsumura, Mamoru Fujii, and Tadao Kasami. On multiple context free grammars. *Theoretical Computer Science*, 88(2):191–229, 1991.

Yos06. Ryo Yoshinaka. Linearization of affine abstract categorial grammars. In *Proceedings of the 11th conference on Formal Grammar*, pages 185–199, 2006.